

# FVPP: Teorijske osnove formalne verifikacije provjerom modela

Vremenska logika CTL

Formalna verifikacija kritičnih programskih dijelova u sustavu NuSMV

**Pripremio: izv. prof. dr. sc. Alan Jović**

Ak. god. 2022./2023.



# Vremenska logika s grananjem

Engl. *Computational Tree Logic*, **CTL**

**M. Ben-Ari**



**E. M. Clarke**



**E.A. Emerson**



# Modalna i vremenska logika

- **Modalna logika** - proširenje klasične logike „modalitetima” istinitosti (subjektivnim konceptima), kao što su: „što mora biti istinito” i „što može biti istinito.”

Primjer:

- $p$  = atomički propozicijski simbol
- Neka je:  $p = F$  (neisitinit) u sadašnjem svijetu (stanju stvari).
- Tada:
- (*moguće*  $p$ ) = T      ako postoji barem jedan drugi svijet (neka druga situacija, neki drugi scenarij, neka druga baza znanja) u kojoj je  $p = T$ .
- (*nužno*  $p$ ) = F      jer (*nužno*  $p$ ) = T akko je  $p$  istinit u svim svjetovima (ovdje F jer je u našem svijetu  $p=F$ ).
- U klasičnu propozicijsku i predikatnu logiku dodaju se **modalni operatori**.
- Modalni opetariori u **vremenskoj logici**: uvijek, konačno, što\_je\_bilo, što\_je\_sada, što\_će\_biti

# Vremenska logika – višestruki pogledi

- **TL propozicijska ili predikatna:** Klasična propozicijska logika proširena vremenskim operatorima. TL prvoga reda nije odlučljiva, ali neki njezine reducirane varijante jesu odlučljive (npr. one do najviše jedne slobodne varijable u predikatnoj formuli, za formule s predikatima od 1 ili 2 varijable).
- **Globalna ili modularna** (endogena ili egzogena). Rasuđivanje o kompletnom sustavu korištenjem vremenskih formula – omogućuje konkurentnost izvođenja, ili samo ograničenje na pojedine dijelove.
- **TL linearnog vremena ili s grananjem vremena:** U svakom trenutku postoji samo jedan budući trenutak (jedna vremenska crta) ili u svakom trenutku može postojati više različitih budućih vremenskih crta.
- **Diskretno ili kontinuirano vrijeme.** U računarstvu uobičajeno diskretno vrijeme (sekvence stanja).
- **Uključuje ili ne uključuje prošlo vrijeme, uz buduće vrijeme.** Izvorno TL obuhvaća oba vremena. U digitalnim sustavima uobičajeni su samo operatori budućeg vremena.
- Trenutačno razmatramo: **TL propozicijsku, globalnu (endogenu), s grananjem vremena, diskretno vrijeme, samo buduće vrijeme:**  
**CTL**

# Kripkeova struktura

**Model sustava koji želimo formalno verificirati :**

= model implementacije (  $I$  )

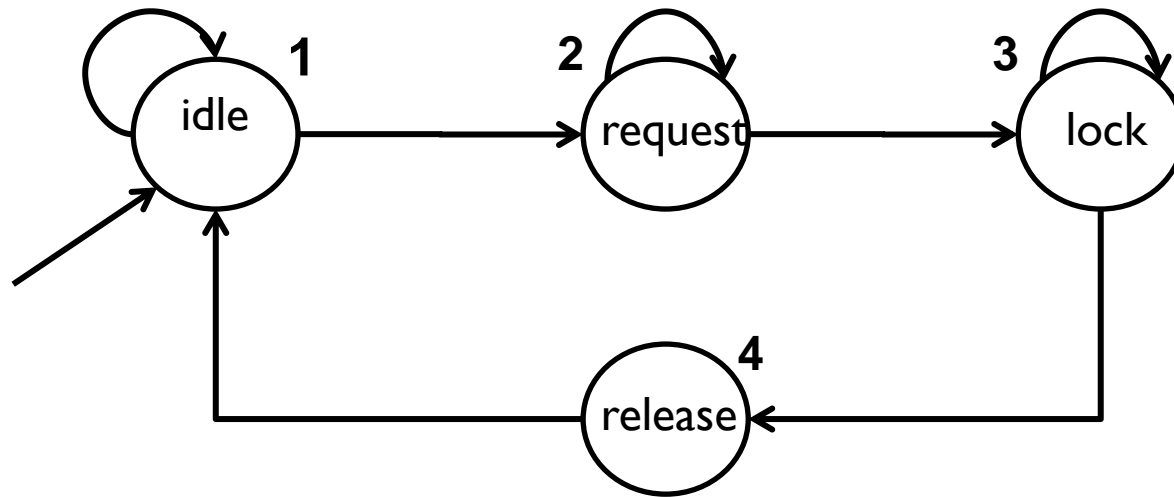
= Kripkeova struktura  $\mathbf{M}$  (Saul Kripke, 1971)

$$I = \mathbf{M} = (\mathbf{S}, \mathbf{R}, \mathbf{L})$$

gdje je :

- $\mathbf{S}$  : konačan skup stanja
- $\mathbf{R} \subseteq \mathbf{S} \times \mathbf{S}$  : totalna binarna relacija (svako stanje je obuhvaćeno; ima sljedbenika), tj.  $\forall s \in \mathbf{S} \ ( \exists t \in \mathbf{S} \mid (s, t) \in \mathbf{R} )$
- $\mathbf{L}$  : funkcija označavanja stanja:  $\mathbf{S} \rightarrow 2^{\mathbf{AP}}$   
**AP**: skup atomičkih propozicijskih simbola  
**L**: (engl. *labeling*) daje interpretaciju svih simbola iz skupa **AP** za svako stanje  $s \in \mathbf{S}$ .

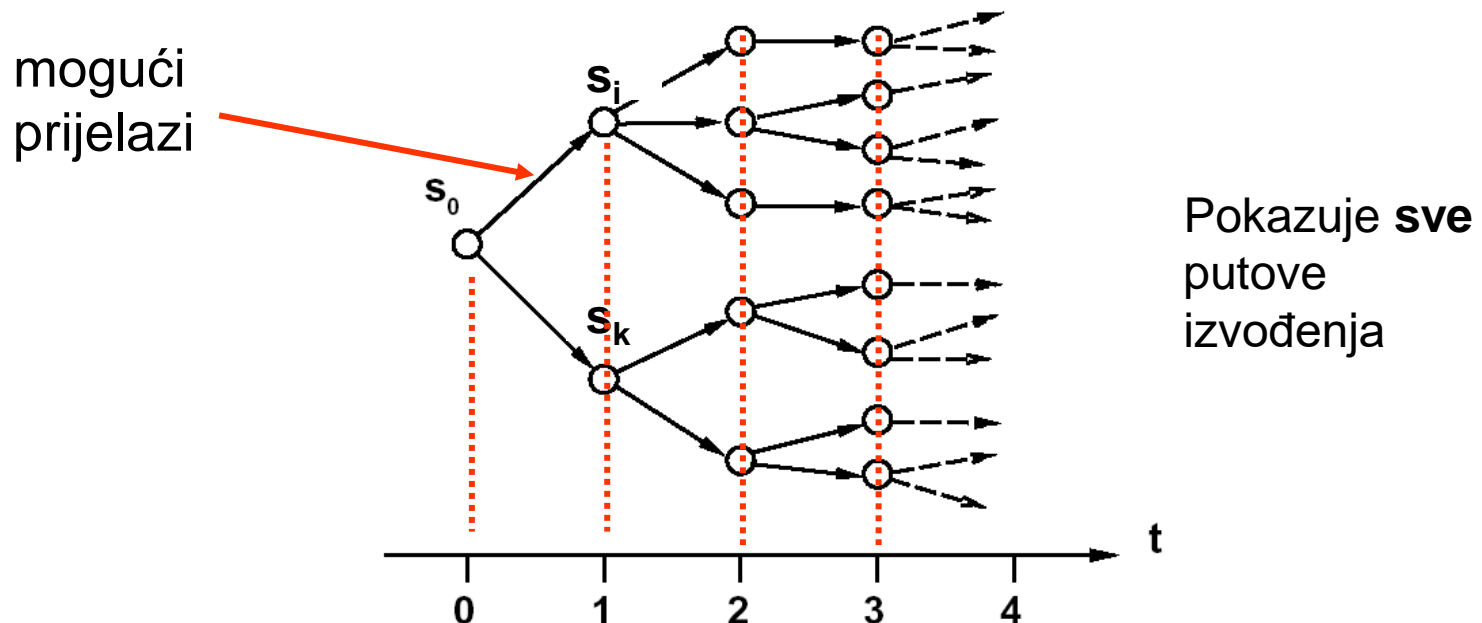
# Primjer Kripkeove strukture



- $S = \{1, 2, 3, 4\}$
- $R = \{(1,1), (1,2), (2,2), (2,3), (3,3), (3,4), (4,1)\}$
- $L: L(1) = \{\text{idle}\}$   
     $L(2) = \{\text{request}\}$   
     $L(3) = \{\text{lock}\}$   
     $L(4) = \{\text{release}\}$
- **Jedan put** (engl. *path*), ili **jedno odvijanje izvođenja sustava** (engl. *one execution, one computation*) je beskonačna sekvenca (slijed) stanja.
- Npr.: 1, 2, 2, 2, 2, 3, 4, 1, 1, ...
- Mogući prijelazi stanja obuhvaćeni su relacijom R.

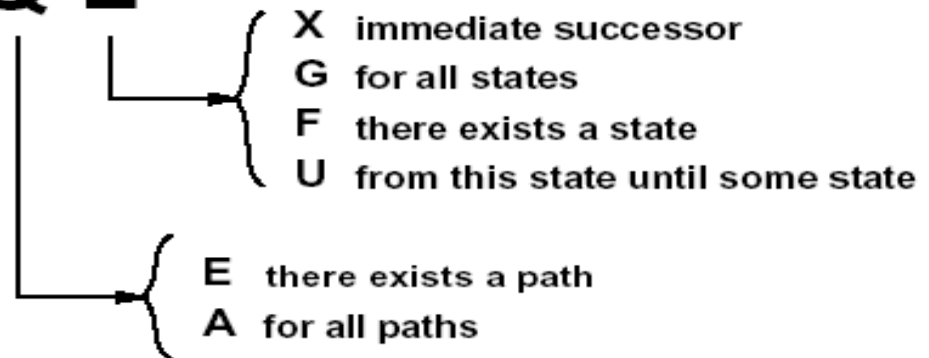
# Vremenska logika s grananjem

- Neka definirana Kripkeova struktura može se promatrati kao beskonačno stablo izvođenja sustava (“odmota” se počevši od promatranog stanja  $s_0$ )
- **To je vremenska logika s grananjem** (engl. *Computation Tree Logic, CTL*)
- Razvijena početkom 1980-tih, koristi se naveliko i danas (razvili: Ben-Ari, Clarke, Emerson i drugi)



# Formula CTL-a (u kontekstu odmotane Kripkeove strukture)

- Kvantifikator puta **Q** i operator stanja **L** čine par **Q L**, koji poprima 8 dozvoljenih kombinacija: **Q L**



**PAZI:**  
Uvijek u  
paru !

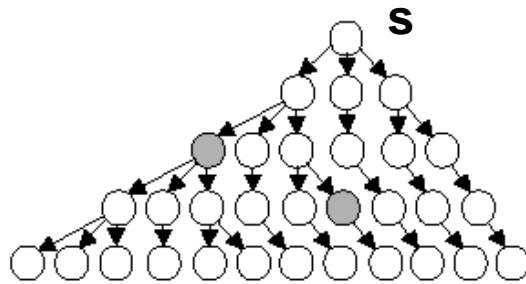
EX	AX
EG	AG
EF	AF
EU	AU



# Semantika formule u CTL-u za model **M** i stanje **S**

## CTL Formulas

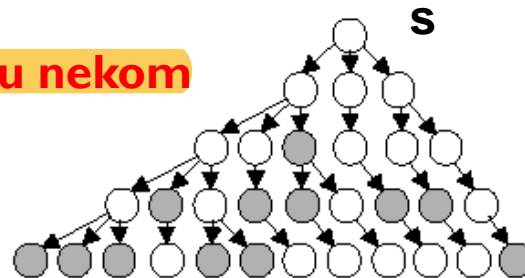
*EF p* “there is a path such that *p* eventually holds”



**EF p = “postoji put (barem jedan) na kojem u nekom budućem stanju vrijedi p”**

*AF p* “all paths are such that *p* eventually holds”

**AF p = “na svim putovima u nekom budućem stanju vrijedi p”**

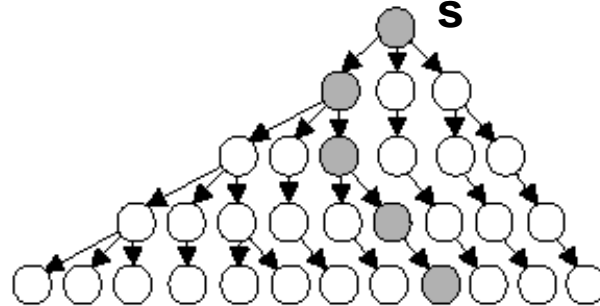


**Napomena: EF p i AF p su istinite ako već i u promatranom (početnom) stanju vrijedi p.**

# Semantika formule u CTL-u za model **M** i stanje **S**

## CTL Formulas Cont.

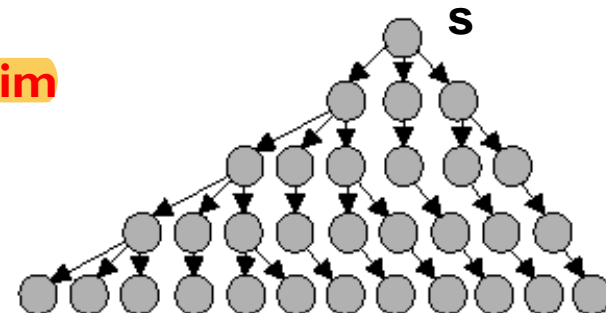
*EG p* “there is a path such that *p* holds invariantly along the path”



**EG p = “postoji put (barem jedan) na kojem u svim budućim stanjima vrijedi p”**

*AG p* “every path is such that *p* holds invariantly along the path”

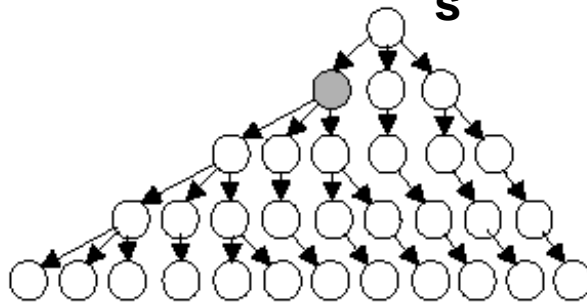
**AG p = “na svim putovima u svim budućim stanjima vrijedi p”**



# Semantika formule u CTL-u za model **M** i stanje **S**

## CTL Formulas Cont.

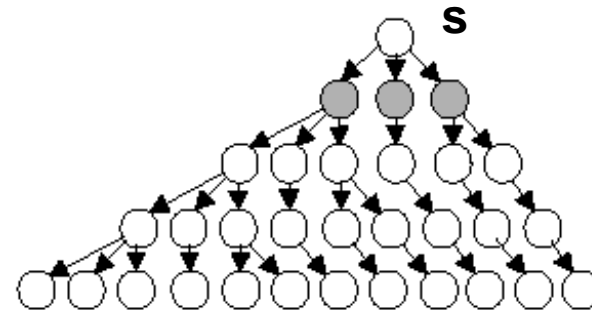
*EX p* “there is a path such that  $p$  holds in the next state along the path”



**EX p = “postoji put (barem jedan) na kojem u sljedećem stanju vrijedi p”**

*AX p* “every path is such that  $p$  holds in the next state along the path”

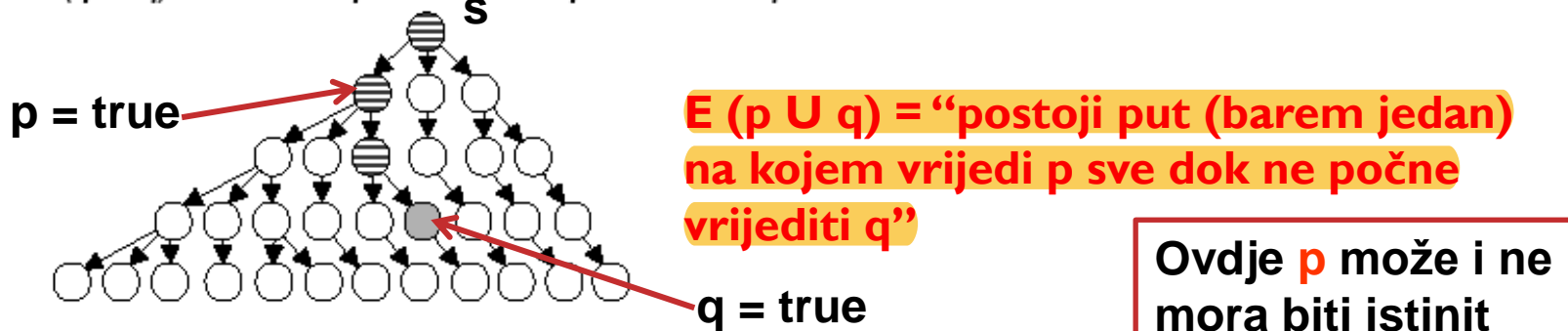
**AX p = “na svim putovima u sljedećem stanju vrijedi p”**



# Semantika formule u CTL-u za model **M** i stanje **S**

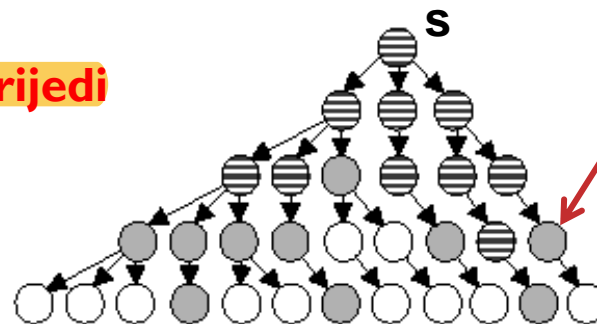
## CTL Formulas Cont.

$E(p \text{ U } q)$  “there is a path such that  $p$  holds until  $q$  holds”



$A(p \text{ U } q)$  “every path is such that  $p$  holds until  $q$  holds”

$A(p \text{ U } q)$  = “na svim putovima vrijedi  $p$  sve dok ne počne vrijediti  $q$ ”



**Napomena:**  $E(p \text{ U } q)$  i  $A(p \text{ U } q)$  su istinite ako već i u promatranom (početnom) stanju vrijedi  $q$  ( $p$  ne mora uopće vrijediti).

# CTL – sintaksa

- AU, EU - binarni operatori
- Ostalih 6 operatora - unarni operatori

## Neke ispravno definirane CTL-formule:

- $AG (q \Rightarrow EG r)$
- $EG p$
- $p \wedge q$
- $E (p \cup q)$
- $A (p \cup EF p)$
- $AG (p \Rightarrow A [p \cup (\neg p \wedge A [\neg p \cup q] ) ] )$

## Neke krivo definirane CTL-formule:

- $FG p$
- $EF (r \cup q)$
- $A[ p \cup (q \cup r) ]$
- $AF [(r \cup q) \wedge (p \cup r)]$

# CTL – semantika

$M = (S, R, L)$  - model sustava (Kripkeova struktura)

- $M, s \models \varphi$  formula vrem. logike  $\varphi$  je istinita u modelu  $M$  za stanje  $s$
- $M, s \not\models \varphi$  formula vrem. logike  $\varphi$  nije istinita u modelu  $M$  za stanje  $s$

1.  $M, s \models p$  istinita

akko  $p \in L(s)$

( $p$  je propozicijski atomički simbol)

2.  $M, s \models (\varphi_1 \wedge \varphi_2)$

akko  $M, s \models \varphi_1$  i  $M, s \models \varphi_2$

3.  $M, s \models (\varphi_1 \vee \varphi_2)$

akko  $M, s \models \varphi_1$  ili  $M, s \models \varphi_2$

4.  $M, s \models (\varphi_1 \Rightarrow \varphi_2)$

akko  $M, s \not\models \varphi_1$  ili  $M, s \models \varphi_2$   
(implikacija)

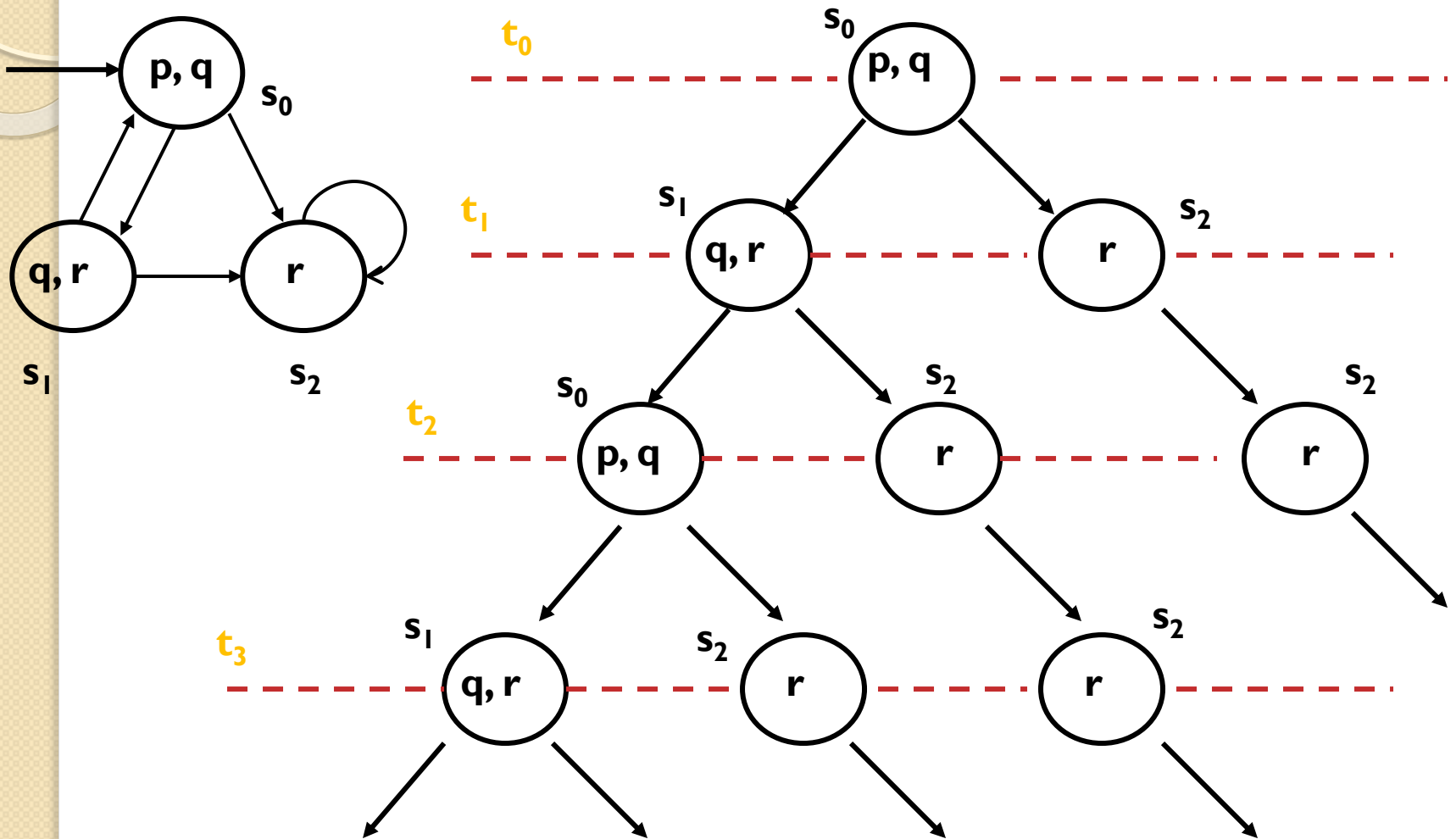
5.  $M, s \models AX \varphi$

akko za sve  $s_i$  takve da  $s \rightarrow s_i$

vrijedi  $M, s_i \models \varphi$

Analogne formalne definicije i za  $EX \varphi, EF \varphi, AF \varphi, EG \varphi, AG \varphi, E(\varphi_1 U \varphi_2)$  i  $A(\varphi_1 U \varphi_2)$

# Primjer “odmotavanja” modela



- $p, q, r$  - propozicijski simboli
- $S_0$  - početno stanje ili stanje koje nas zanima

# Neke istinite CTL-formule za model s prethodnog slajda

- $M, s_0 \models (p \wedge q)$  ; atomi  $p$  i  $q$  su istiniti u stanju  $s_0$
- $M, s_0 \models \neg r$  ; atom  $r$  nije istinit u stanju  $s_0$
- $M, s_0 \models EX (q \wedge r)$  ; postoji put gdje je za sljedeće stanje vrijedi  $(q \wedge r)$
- $M, s_0 \models \neg AX (q \wedge r)$  ; postoji jedan put na kojem ne vrijedi za sljedeće stanje  $(q \wedge r)$
- $M, s_0 \models \neg EF (p \wedge r)$  ; nema puta sa stanjem za koje vrijedi  $(p \wedge r)$
- $M, s_0 \models AF r$  ; duž svih putova možemo dosegnuti stanje za koje vrijedi  $r$
- $M, s_0 \models E [(p \wedge q) U r]$  ; postoji put iz  $s_0$  na kojem u svim stanjima  $(p \wedge q) = \text{True}$ , dok  $r = \text{True}$  (npr. do  $s_2$ )
- $M, s_0 \models A [p U r]$  ; na svim putovima vrijedi  $[p U r]$



# Ekvivalentne CTL-formule i adekvatni skup

$\neg \mathbf{AF} \varphi$	$=$	$\mathbf{EG} \neg \varphi$	<b>; de Morgan</b>
$\mathbf{AF} \varphi$	$=$	$\neg \mathbf{EG} \neg \varphi$	
$\mathbf{EG} \varphi$	$=$	$\neg \mathbf{AF} \neg \varphi$	
$\mathbf{AG} \neg \varphi$	$=$	$\neg \mathbf{EF} \varphi$	<b>; de Morgan</b>
$\mathbf{AG} \varphi$	$=$	$\neg \mathbf{EF} \neg \varphi$	
$\neg \mathbf{AX} \varphi$	$=$	$\mathbf{EX} \neg \varphi$	<b>; X je vlastiti dual</b>
$\mathbf{AX} \varphi$	$=$	$\neg \mathbf{EX} \neg \varphi$	
$\mathbf{AF} \varphi$	$=$	$\mathbf{A} (\text{True} \mathbf{U} \varphi) = \neg \mathbf{EG} \neg \varphi$	
$\mathbf{EF} \varphi$	$=$	$\mathbf{E} (\text{True} \mathbf{U} \varphi)$	
$\mathbf{EG} \varphi$	$=$	$\neg [\mathbf{A} [\text{True} \mathbf{U} \neg \varphi]]$	

- **Ponekad notacija:**  
$$\begin{aligned} \mathbf{A}[p \mathbf{U} q] &= [p \mathbf{AU} q] \\ \mathbf{E}[p \mathbf{U} q] &= [p \mathbf{EU} q] \end{aligned}$$
- Temeljem gornjih ekvivalencija, za izračun svih CTL-formula dovoljno je imati postupke za izračun EX, EG, EU = **adekvatni skup** (engl. *adequate set*). Postoji više takvih adekvatnih skupova.

# Primjeri preslikavanja prirodnog jezika u CTL-formule – specifikacija ponašanja

1. „Moguće je doći u stanje gdje  $\text{start}=\text{T}$  i  $\text{ready}=\text{F}$ .”

**EF ( $\text{start} \wedge \neg \text{ready}$ )**

2. „Za svako stanje, ako se postavi zahtjev (npr. za nekim resursom), onda će on biti konačno prihvaćen (kad-tad).”

**AG ( zahtjev  $\Rightarrow$  AF prihvaćen)**

3. „U svakom slučaju, određeni proces će nakon konačnog vremena biti stalno zaustavljen.”

**AF (AG zaustavljen)**

4. „Iz svakog stanja moguće je barem na nekom putu doći do stanja "restart".”

**AG (EF restart)**

# Primjeri preslikavanja prirodnog jezika u CTL-formule – specifikacija ponašanja

5. „Kadgod je  $in = 1$ , nakon dva takta je uvijek  $out = 1$ .“

**$AG (in = 1 \Rightarrow AX AX out = 1)$**

6. „Uvijek vrijedi: ako se pojavi "send" onda konačno "receive" postaje istinit, te do tog trenutka "send" mora ostati istinit.“

**$AG (send \Rightarrow A(send \ U \ receive))$**

# Primjena CTL-a u provjeri modela (engl. *CTL model checking*)

- Za danu Kripkeovu strukturu (usmjereni označeni graf) i određen skup početnih stanja  $S_0$ , provjeri da CTL-formula vrijedi za ta stanja.
- Formalno:  $M, S_0 \models \phi$  , tj.  $\forall s_0 \in S_0 \quad M, s_0 \models \phi$  (za svako stanje iz  $S_0$ )

## Postupak

- Potrebno je pronaći sva stanja koja zadovoljavaju CTL-formulu  $\phi$  i ispitati da li je željeni podskup  $S_0$  uključen.
- Provjera modela u CTL-u povlači **rukovanje skupovima stanja**.

# Rješavanje ugniježđenih CTL- formula

- Primjenjuje se induktivno pravilom “iznutra prema van”
- Računa se skup stanja koja zadovoljavaju najviše unutarnju CTL-formulu
- Koriste se ti rezultati za izračun sve više vanjskih formula

## Primjer (UNESCO math&dev TUNIS 2008):

Za zadanu Kripkeovu strukturu

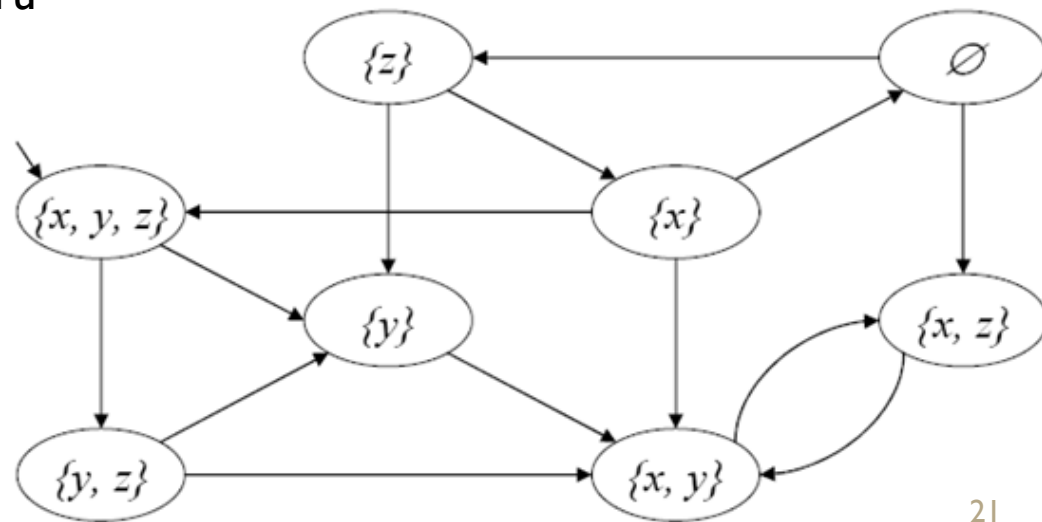
izračunaj skup stanja  $S_k$

koja zadovoljavaju CTL

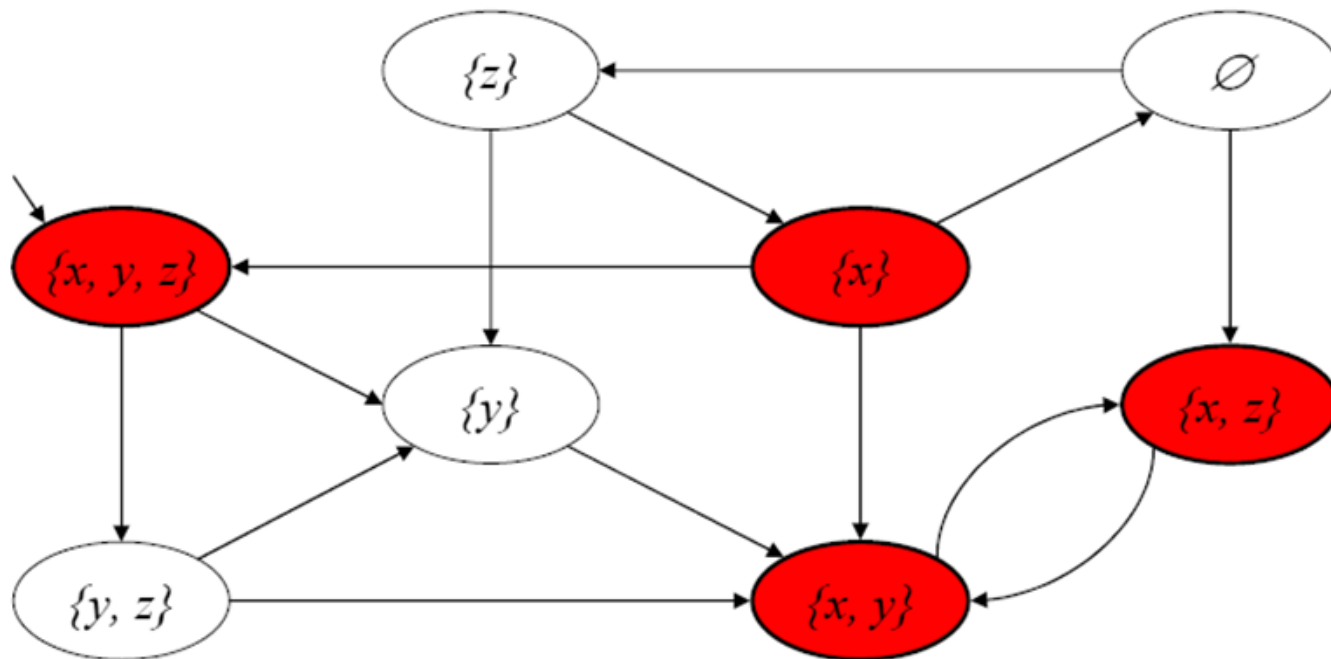
formulu  $AF\ AG(x)$

Postupak:

1. Izračunaj  $S_k(x)$
2. Izračunaj  $S_k(AG(x))$
3. Izračunaj  $S_k(AF(AG(x)))$

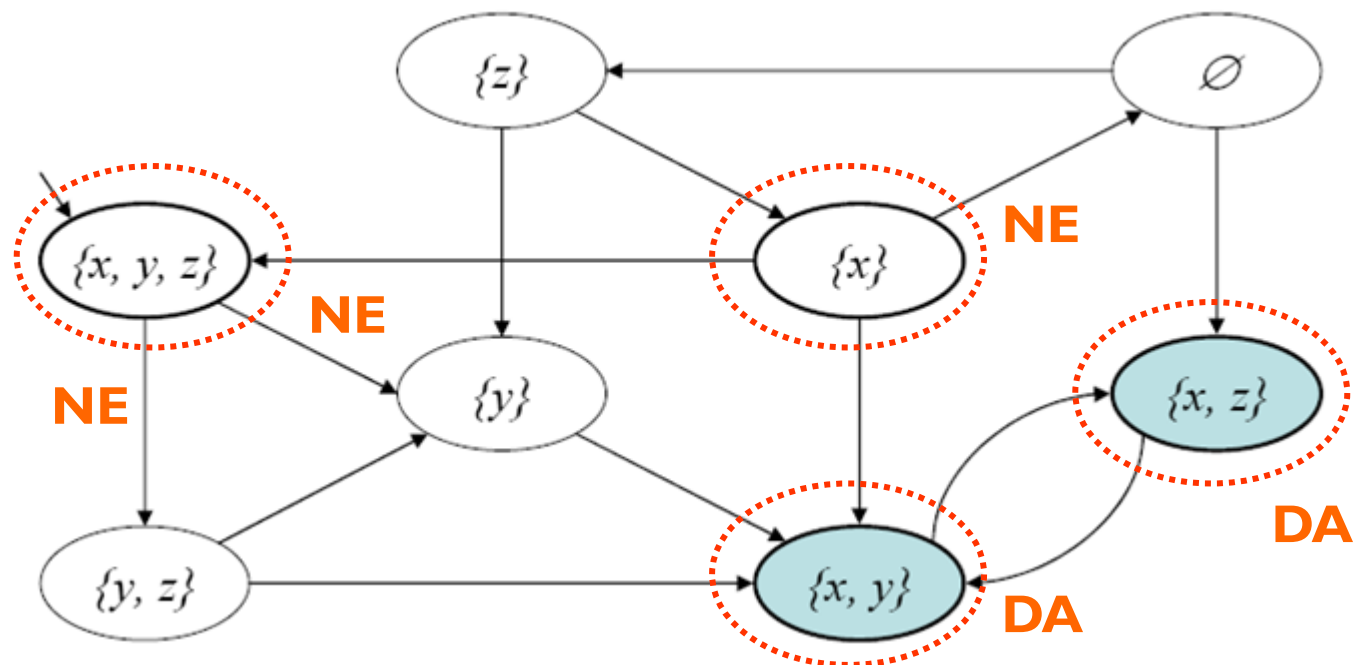


Skup stanja koja zadovoljavaju  
formulu  $x = S_k(x)$ :



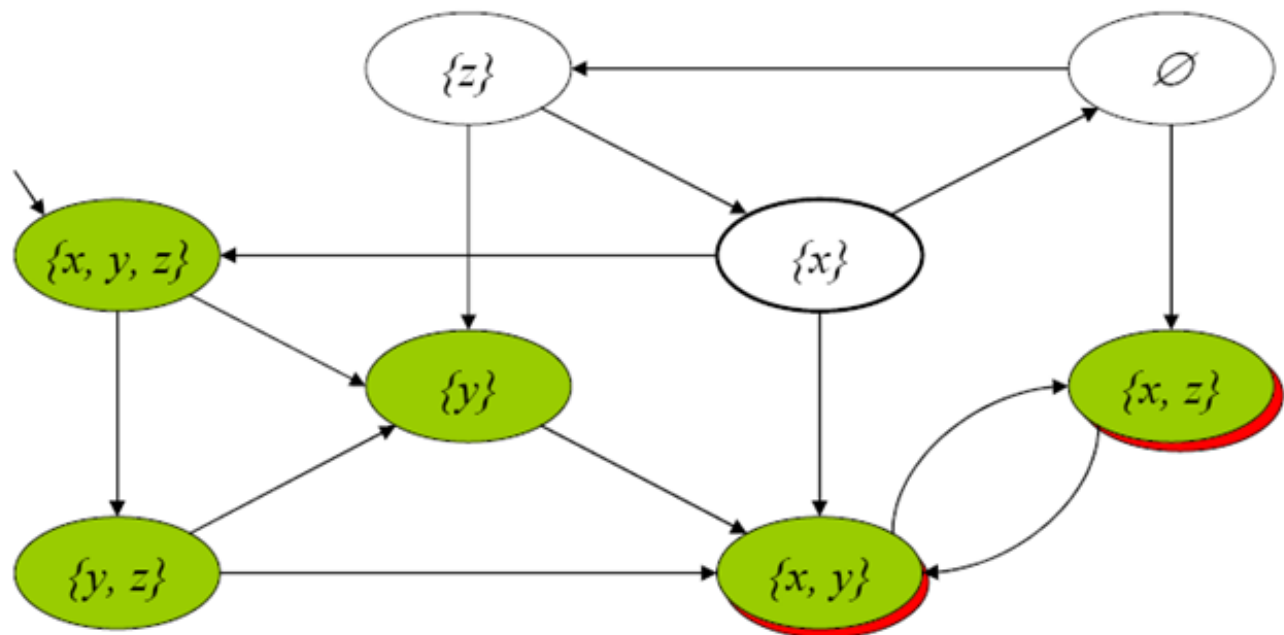
# Skup stanja koja zadovoljavaju formulu $AG(x) = S_k(AG(x))$ :

- Da bi se zadržalo, stanje koje sadrži  $x$  mora biti korijen prijelaza koja dalje sva sadržavaju  $x$  – trebaju se analizirati prethodnici stanja u kojima vrijedi  $x$



# Skup stanja koja zadovoljavaju formulu $AF(AG(x)) = S_k(AF(AG(x)))$ :

- Tražimo prethodnike prošla dva stanja (koja zadovoljavaju  $AG(x)$ ) koji na svim svojim putevima konačno dolaze do ta dva stanja, uklone se oni prethodnici za koje to ne vrijedi.

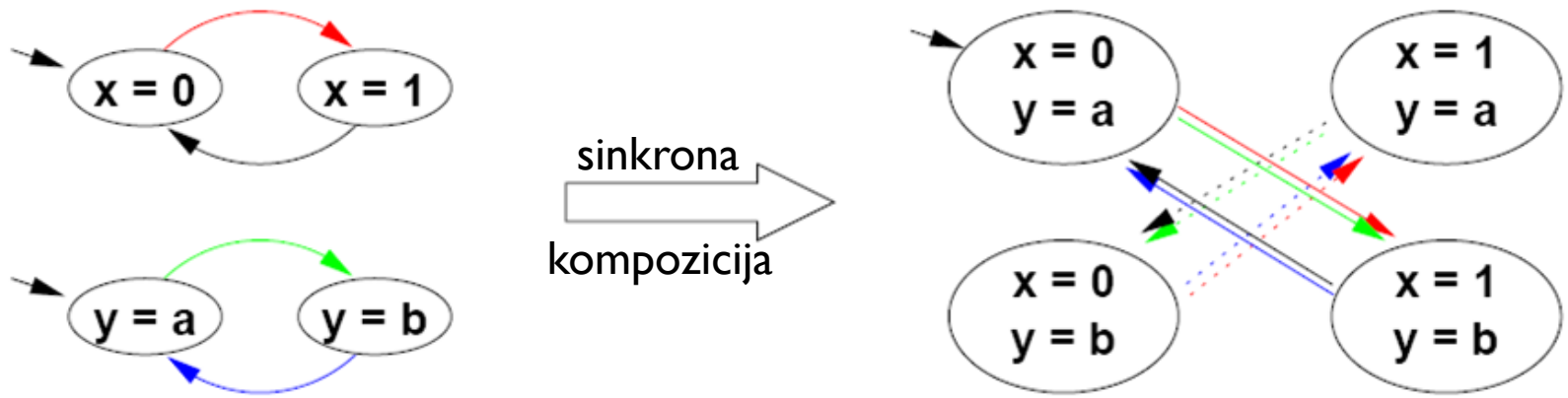




# Sinkrona i asinkrona kompozicija Kripkeovih struktura

- Složeni Kripkeovi modeli tipično se grade od jednostavnih.
- **Sinkrona kompozicija: u svakom vremenskom trenutku sve komponente mijenjaju stanje** (istodobno, paralelno).
- Tipičan primjer: **sekvencijsko sklopovlje**

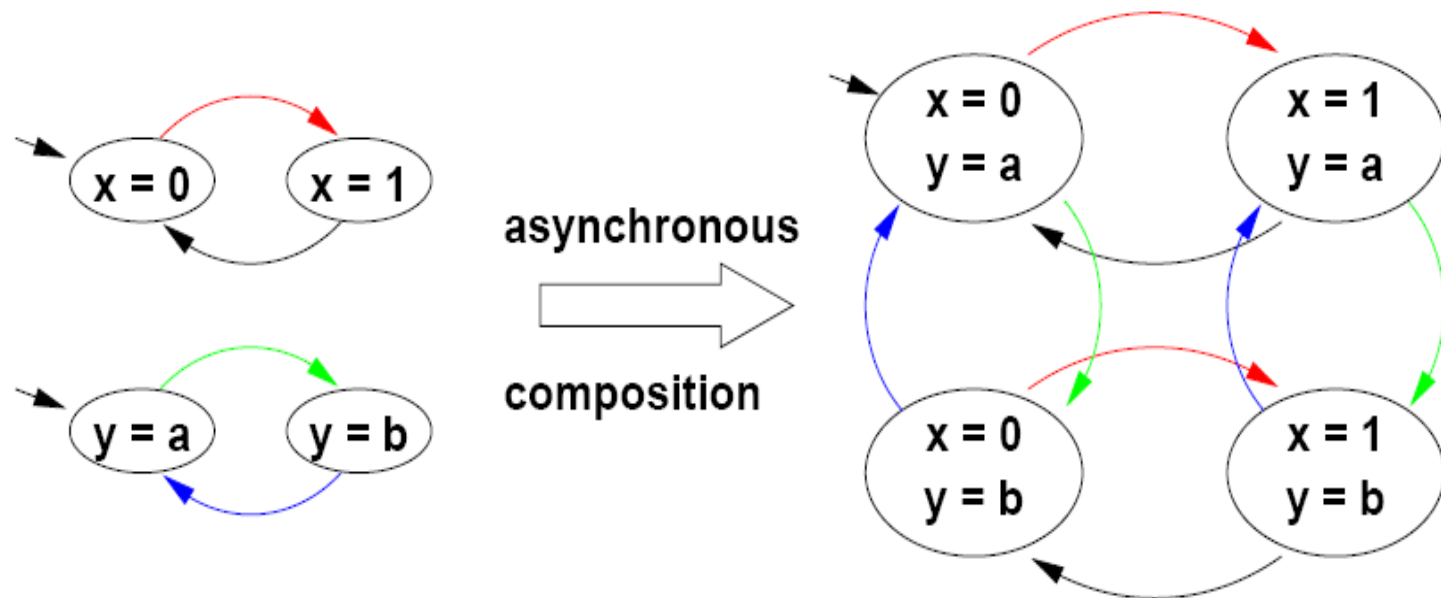
(pratiti boju strelica):



# Sinkrona i asinkrona kompozicija Kripkeovih struktura

- **Asinkrona kompozicija:** u svakom vremenskom trenutku jedna komponenta mijenja stanje.
- Tipičan primjer: **komunikacijski protokol.**

(pratiti boju strelica):



# Zadaci

1. Koje su od navedenih sintaksno dobro definirane formule u logici CTL:

- a)  $p \wedge \neg q \vee EX p$
- b)  $AF (G p \Rightarrow GF q)$
- c)  $AG AF EX (p \wedge q)$
- d)  $A((p U q) \vee (q U \neg r))$
- e)  $E(p \Rightarrow E(q \Rightarrow r))$
- f)  $A ((q \wedge r) U (p \wedge r))$

2. Preslikajte rečenice prirodnog jezika u formule logike CTL:

- a) “Uvijek u svim stanjima sustava vrijedi da p ne vrijedi dok ne počne vrijediti q.”
- b) “Postoji put u kojem se konačno dolazi do stanja u kojem vrijedi p i od kojeg dalje p ne vrijedi u sljedeća dva stanja.”
- c) “U početnom stanju vrijedi p, a zatim u sljedećem stanju barem na jednom putu izvođenja ne vrijedi q.”



Formalna verifikacija kritičnih dijelova programa

**Sustav NuSMV**

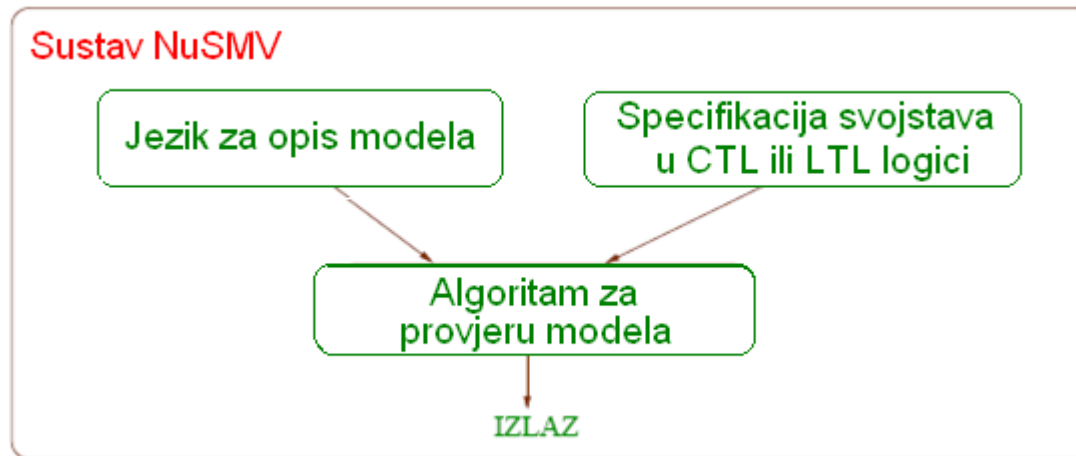
# Uvod

- NuSMV
  - Sustav za simboličku provjeru modela (engl. *symbolic model checking*)
  - Nadograđena inačica izvornog sustava SMV (otuda ono Nu [nju] ☺)
  - Trenutna verzija NuSMV 2.6.0
- Izvorni SMV (engl. *Symbolic Model Verifier*)
  - Razvio ga je Ken McMillan, CMU, u doktoratu iz 1992.



# Uvod

- Formalno: program koji provjerava vrijedi li:  
 $M \models \phi$ , (implementacija  $M$  logički zadovoljava specifikaciju  $\phi$ ) pri čemu je  $M$  Kripke struktura,  $\phi$  je svojstvo izraženo u vremenskoj logici (CTL-u ili LTL-u).
- NuSMV: verifikacija kritičnih dijelova programskih produkata



# Ulaz i izlaz sustava

- **Ulaz sustava:**
  - Tekstualni opis programa u jeziku NuSMV - model sustava
  - Specifikacija svojstava u CTL ili LTL logici
    - Postoji i mogućnost specifikacije u RTCTL (Real-Time CTL)) i PSL-u - Property Specification Language)
  - U sklopu ovog kolegija na laboratorijskim vježbama koristit ćemo samo CTL logiku pri specifikaciji i provjeri svojstava u NuSMV-u
- **Izlaz sustava:**
  - 'TRUE' - ako je specifikacija zadovoljena za sva početna stanja modela
  - 'FALSE' - inače + ispis (engl. *trace*) koji pokazuje zašto navedeno svojstvo nije ispunjeno u modelu opisanim NuSMV programom

# Algoritam za provjeru modela

- Algoritam se temelji na simboličkom prikazu stanja i prijelaza pomoću:
  - **BDD-ova** – binarnih dijagrama odlučivanja
    - Za logike CTL i LTL, verifikacija beskonačnih putova
  - **SAT-rješavača**
    - Za logiku LTL, verifikacija konačnih putova - ograničena provjera modela (engl. *Bounded model checking*)
- Teme predavanja koja slijede nakon međuispita



# NuSMV: ulazni jezik

- Motivacija prilikom modeliranja
  - opis upravljanja sustavom i interakcija dijelova sustava
  - posebno služi modeliranju **kritičnih programskih dijelova**
  - posebno značajno za modeliranja ponašanja višedretvenih programa prilikom čega može doći do više vrsta problema (utrka, izgladnjivanje, potpuni zastoj, itd.)
- Namjena ulaznog jezika: opis relacije prijelaza (tranzicijske relacije) konačne Kripkeove strukture
  - Pomoću sustava NuSMV opisuju se međuovisni strojevi s konačnim brojem stanja (engl. *Communicating Finite State Machines*).
  - Jezik NuSMV ne podržava složene strukture podataka (liste, stabla, mape...)
- Sličan jezicima za opis sklopovlja, ali mu to nije namjena

# NuSMV: programi

- Programi u NuSMV-u sastoje se od jednog ili više modula
  - Jedini posebni modul je **main** (slično kao C, Java, ...)
- Svaki modul se obično sastoji od:
  - Deklaracije varijabli
  - Dodjeljivanja vrijednosti varijablama
  - Svojstava koja se žele provjeriti (uglavnom u modulu main)

# Variable (1/2)

- Variable u NuSMV-u predstavljaju **variable stanja** modela i instance pojedinih modula
- Deklaracija varijabli:  

```
<decl> ::=  
  "VAR"  
  atom1 ":" <type> ";"  
  ...
```
- Tip varijable (<type>) može biti:
  - **boolean**
  - **riječ (vektor bitova određene duljine)**
  - **enumerirani (pobrojani) tip**
  - **konačan podniz cijelih brojeva**
  - **korisnički definirani modul (instanca modula),**
  - **niz prethodno navedenih tipova**

# Variable (2/2)

- Neki primjeri deklaracija:

**VAR**

*a : boolean;*

*w : word[7];*

*switch : {on, off};*

*state : {start, request, busy, stop};*

*n : 1..10000;*

*server6 : Server(req, stateClient);*

*producerWG : process Producer();*

*arr : array 1..20 of {start, busy, stop};*

- Tip **boolean** može poprimiti vrijednosti **FALSE** ili **TRUE** (paziti: od verzije NuSMV 2.5.1 više boolean vrijednosti nisu 0 ili 1!!!)
- Ključna riječ **process** ispred imena modula definira asinkroni način izvođenja

# Dodjeljivanja (1/4)

- Dodjeljivanje (engl. *assignment*) daje varijabli:
  - Trenutačnu vrijednost
  - Inicijalnu vrijednost - *init*
  - Sljedeću vrijednost (na temelju trenutnih vrijednosti varijabli programa) - *next*
- Dodjeljivanje vrijednosti varijablama:

```
<assignment> ::=  
  "ASSIGN"  
  <dest> " := " <expr> ";"  
  ...  
  <dest> ::=  
    atom  
    | "init" "(" atom ")"  
    | "next" "(" atom ")"
```

# Dodjeljivanja (2/4)

- Na lijevoj strani dodjeljivanja može se naći:
  - **atom** označava trenutnu vrijednost varijable,
  - ***init*(atom)** označava početnu vrijednost varijable,
  - ***next*(atom)** označava vrijednost varijable u sljedećem stanju
- Izraz (**<expr>**) na desnoj strani dodjeljivanja može se evaluirati u:
  - **Cjelobrojnu vrijednost ili simboličku konstantu**
  - **Skup vrijednosti**

U prvom slučaju lijeva strana jednostavno postaje desna, a u drugom lijeva strana postaje jedan od elemenata s desne strane (**nedeterminizam**)

- Primjeri dodjela:

***ASSIGN***

***a := n mod 2;***

***init(switch) := off;***

***next(state) := {busy, stop};***

# Dodjeljivanja (3/4)

- Ograničenja na dodjeljivanja

1. Svakoј varijabli dodjeljivanje može biti provedeno samo jednom i to ili u “init” i “next” bloku ili za trenutачnu vrijednost, npr:

- `init(status) := busy;` nedozvoljeno

- `init(status) := stop;`

- `next(status) := busy;`

- `next(status) := stop;` nedozvoljeno

- `next(status) := busy;`

- `status := stop;` nedozvoljeno

- `init(status) := busy;`

- `next(status) := stop;` dozvoljeno

# Dodjeljivanja (4/4)

- Ograničenja na dodjeljivanja

2. Varijabla ne može imati cikluse u svojem grafu ovisnosti koji nisu odvojeni s kašnjenjem, npr:

- $x := (x + 1) \bmod 2;$  nedozvoljeno
- $next(x) := x \ \& \ next(x);$  nedozvoljeno
- $next(x) := x \ \& \ next(y);$  nedozvoljeno
- $next(y) := y \ \& \ next(x);$
- $next(x) := x \ \& \ next(y);$  dozvoljeno
- $next(y) := y \ \& \ x;$



# Specifikacija svojstava (1/2)

- Specifikacija ponašanja sustava zadana je formulom u vremenskoj logici CTL
- Zapis CTL-formula u sustavu NuSMV :

`<form> ::= "CTLSPEC" <ctlform> -- osim CTLSPEC može i SPEC`

`<ctlform> ::=`

`<expr> -- Booleova formula`

`| "!" <ctlform> -- logička negacija`

`| <ctlform1> "&" <ctlform2> -- logički i`

`| <ctlform1> "|" <ctlform2> -- logički ili`

`| <ctlform1> "->" <ctlform2> -- logička implikacija`

`| <ctlform1> "<->" <ctlform2> -- logička ekvivalencija`

`| "E" <pathform> -- egzistencijalni kvantifikator puta`

`| "A" <pathform> -- univerzalni kvantifikator puta`

# Specifikacija svojstava (2/2)

- Zapis CTL-formula u sustavu NuSMV (nastavak):

***<pathform> ::=***

***"X" <ctlform> -- operator sljedećeg stanja***  
***| "F" <ctlform> -- operator konačnog dolaska u stanje***  
***| "G" <ctlform> -- operator cijelog puta***  
***| "[" <ctlform1> "U" <ctlform2> "]" -- operator kod kojeg***  
***na cijelom putu ctlform1 vrijedi dok ne počne ctlform2***  
***vrijediti***

- ***Svaka CTL-formula definira se novom ključnom riječi CTLSPEC***
- Primjeri specifikacija:  
***CTLSPEC AG AF state=start***  
***CTLSPEC EF n>=40***  
***CTLSPEC E [ b=TRUE U state=busy ]***  
***CTLSPEC AG (state=request -> AF state=busy)***

# NuSMV: primjer glavnog programa

**MODULE** main

**VAR**

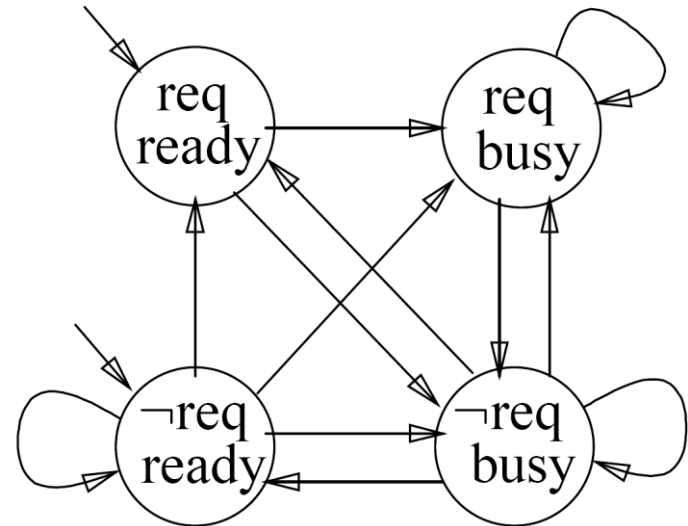
```
request : boolean;  
status : {ready,busy};
```

**ASSIGN**

```
init(status) := ready;  
next(status) := case  
    request : busy;  
    TRUE : {ready,busy};  
esac;
```

**CTLSPEC**

```
AG (request -> AF status =  
    busy)
```



FSM se sastoji od četiri stanja - svako stanje odgovara mogućoj vrijednosti dviju binarnih varijabli

Variable:

*request* - varijable tipa boolean, TRUE ili FALSE

*status* - je enumerirana varijabla koja može poprimiti vrijednost ready ili busy

Inicijalna i sljedeće vrijednosti varijable *request* nisu određene u programu. Na taj je način modeliran utjecaj okoline (NuSMV sustav može pridijeliti proizvoljne vrijednosti toj varijabli).

Varijabla *status* je parcijalno određena: inicijalno poprima vrijednost *ready*, i postaje *busy* ako je *request* jednak *TRUE*. Inače ako je *request* jednak *FALSE* vrijednost varijable nije određena.

# Nedeterminističke vrijednosti

- Varijabla može nedeterministički poprimiti vrijednost na dva osnovna načina:
  - **Implicitno** – varijabli se ne dodjeljuje nikakva vrijednost (ulazna varijabla)
  - **EksPLICITNO** – nedeterminističkim dodjelama: varijabli se dodjeljuje skup vrijednosti (varijabla nedeterministički poprima jednu vrijednost iz skupa)
- Prethodni primjer implicitnog nedeterminizma
  - Utjecaj varijable *request* primjer je implicitnog nedeterminizma
- Primjer eksplicitnog nedeterminizma :
  - ***next(status) := {ready, busy};***
- Nedeterminističke dodjele – koriste se za modeliranje utjecaja okoline, nepotpune implementacije ili za formiranje apstrakcija složenih protokola, gdje se ne može utvrditi točna vrijednost varijable stanja

# Moduli u NuSMV-u

- NuSMV podržava opis sustava razbijanjem na više modula (omogućuje provjeru svojstava interakcije između dijelova sustava)
- Modul se instancira tako da se deklarira varijabla čiji je tip ime modula
- Pristup varijablama unutar modula obavlja se pomoću točke "."
  - ***m.v - pristupa se varijabli v unutar modula m***

# Primjer: Trobitni brojač

```
MODULE main
```

```
VAR
```

```
    bit0 : counter_cell(TRUE) ;
```

```
    bit1 : counter_cell(bit0.carry_out) ;
```

```
    bit2 : counter_cell(bit1.carry_out) ;
```

```
CTLSPEC
```

```
    AG AF bit2.carry_out
```

```
MODULE counter_cell(carry_in)
```

```
VAR
```

```
    value : boolean;
```

```
ASSIGN
```

```
    init(value) := FALSE;
```

```
    next(value) := value xor carry_in;
```

```
DEFINE
```

```
    carry_out := value & carry_in;
```

# NuSMV primjer: Trobitni brojač

- Ovaj program opisuje brojač koji uzastopno broji od 000 do 111
- Trobitni brojač je opisan pomoću tri jednobitna brojača:
  - Modul `counter_cell` je instanciran tri puta: `bit0`, `bit1` i `bit2`.
  - Svaki modul ima jedan formalni parametar: `carry_in`
  - Moduli su međusobno povezani tako da je `carry_in` modula `bit1` ujedno `carry_out` modula `bit0` (analogno vrijedi za `bit2` i `bit1`)
- Ključna riječ **DEFINE** se koristi kao makro za dodjelu izraza `value & carry_in` simbolu `carry_out`
- Značenje ključne riječi **DEFINE** je to da gdje god da se dalje u programu pojavi `carry_out`, bit će zamijenjen s izrazom `value & carry_in`
- Ključna riječ **DEFINE** ne proširuje prostor stanja
- **DEFINE** ne može raditi s nedeterminističkim vrijednostima

# Sinkrono i asinkrono izvršavanje u NuSMV-u

- **Sinkroni način izvršavanja** - postoji globalni sat, i svakim taktom globalnog sata, svaki od modula (ASSIGN blok) se izvršava paralelno
- **Asinkroni način izvršavanja** - pojedini dijelovi sustava (moduli) odvijaju se različitim brzinama, pri čemu se njihova izvođenja proizvoljno isprepliću (engl. *interleaving execution*)
  - **U svakom taktu, proizvoljno se odabire jedan od modula i izvršava se – unutarnja varijabla *running* za taj modul se postavlja na TRUE)**
  - **Ukoliko se želi da se neki modul izvršava u *interleaved* načinu izvođenja potrebno je postaviti ključnu riječ *process* prilikom instanciranja modula**
  - **Korisno za opis komunikacijskih protokola, asinkronih sklopova i ostalih sustava koji nisu sinkronizirani pomoću nekog globalnog sata te za modeliranje višedretvenosti**



# Međusobno isključivanje (engl. *mutual exclusion*, MUTEX)

- Ako konkurentni procesi (procesi koji se istovremeno izvode) dijele određeni resurs (datoteka na disku, zapis u bazi podataka, podatak u zajedničkoj memoriji), nužno je osigurati da procesi ne mogu istovremeno pristupati dijeljenom resursu
- Potrebno je definirati *kritične odsječke* unutar svakog od procesa i osigurati da samo jedan proces smije istovremeno biti u kritičnom odsječku
- Problem: pronalaženje protokola kojim se utvrđuje kad koji od procesa smije ući u svoj kritični odsječak

# Svojstva protokola MUTEX

## **Sigurnost (engl. *safety*)**

- Protokol osigurava da samo jedan proces bude u kritičnom odsječku u bilo kojem trenutku

## **Životnost (engl. *liveness*)**

- Kada god neki od procesa želi ući u kritični odsječak, uvijek će konačno i ući

## **Svojstvo neblokiranja (engl. *non-blocking*)**

- Proces uvijek može zahtijevati ulazak u kritični odsječak, drugi ga proces ne smije u tome sprečavati

## **Svojstvo nedeterminiranog redoslijeda (engl. *no strict sequencing*)**

- Procesi mogu ulaziti u svoje kritične odsječke na proizvoljan način (ne postoji predodređeni redoslijed ulaska u kritični odsječak)

# Primjer protokola MUTEX u NuSMV-u (1/3)

```
MODULE main
```

```
VAR
```

```
    pr1: process proc(pr2.st, turn, FALSE);
```

```
    pr2: process proc(pr1.st, turn, TRUE);
```

```
    turn: boolean;
```

```
ASSIGN
```

```
    init(turn) := FALSE;
```

```
    ...
```

- Varijabla `turn` određuje koji proces smije ući u kritični odsječak
- Dvije instance modula `proc`, koji opisuje kontrolne aspekte konkurentnih procesa

# Primjer protokola MUTEX u NuSMV-u (2/3)

...

**MODULE** `proc(other-st, turn, myturn)`

**VAR**

`st` – interno stanje procesa

`st: {n, t, c};`

[n]on-critical – proces nije u kritičnom odsječku

**ASSIGN**

[t]rying – proces želi ući u kritični odsječak

`init(st) := n;`

[c]ritical – proces je u kritičnom odsječku

`next(st) :=`

Ako je proces u nekritičnom odsječku, može u njemu ostati ili preći u stanje `t` (trying – želi ući u kritični odsječak)

**case**

`(st = n) : {t, n};`

Ako je proces u stanju `t`, a drugi proces je u nekritičnom odsječku, tada proces može ući u kritični odsječak (preći u `c`)

`(st = t) & (other-st = n) : c;`

`(st = t) & (other-st = t) & (turn = myturn) : c;`

`(st = c) : {c, n};`

Ako je proces u stanju `t`, i drugi proces je također u tom stanju, tada proces može ući u kritični odsječak (preći u stanje `c`) samo ako je njegov red (`turn=myturn`)

`TRUE : st;`

**esac;**

...

Ako je proces u stanju `c` (kritični odsječak) onda može ostati u tom stanju ili izaći iz kritičnog odsjeka (preći u stanje `n`).

# Primjer protokola MUTEX u NuSMV-u (3/3)

```
...
next(turn) :=
  case
    turn = myturn & st = c : !turn;
    TRUE : turn;
  esac;
JUSTICE running
JUSTICE !(st = c)
```

- Ako je proces u kritičnom odsječku i varijabla **turn** je pokazivala da je njegov red za izvođenje tada se negira njena vrijednost i prebacuje pravo na drugi proces.
- Varijabla **turn** određuje koji će proces ući u kritični odsječak ako u jednom trenutku oba procesa žele ući u kritični odsječak (**st=t**).

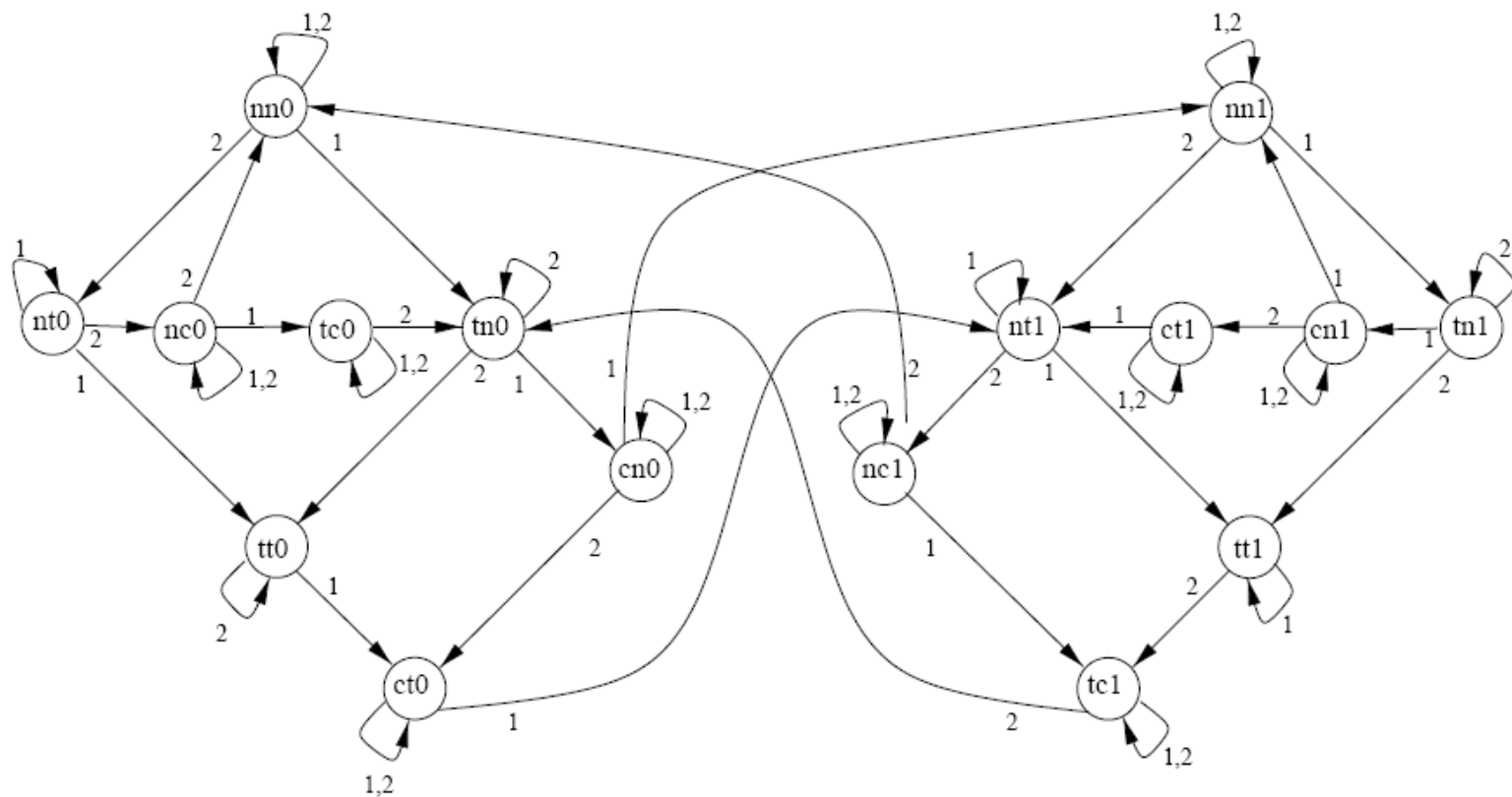
# Pravednost (engl. *fairness*) u NuSMV-u

- Važno svojstvo sustava NuSMV:
  - moguće je ograničiti područje pretraživanja prostora stanja na samo one staze izvršavanja duž kojih je proizvoljna CTL formula istinita beskonačno često
- Obično te staze izvršavanja modeliraju **pravedan** (engl. *fair*) pristup određenom resursu (preciznije, radi se o Büchiovom tipu pravednosti).
- Ograničenje pravednosti sastoji se od:
  - Ključne riječi **JUSTICE** (može i **FAIRNESS**)
  - **CTL formule  $f$**
- Prilikom provjere nekog svojstva, NuSMV će ignorirati svaku stazu na kojoj formula  **$f$**  ne vrijedi beskonačno često.
- **JUSTICE running** – označava da se svaki proces treba izvršavati beskonačno često

# Pravednost u primjeru MUTEX

- **JUSTICE running**
  - Ako je modul koji se razmatra deklariran pomoću ključne riječi *process*, tada će u svakom trenutku NuSMV nedeterministički odrediti hoće li se taj modul izvršavati ili ne. U ovom slučaju razmatrat će se samo ona izvršavanja za koja vrijedi da će modul biti beskonačno često izabran za izvođenje.
- **JUSTICE ! (st = c)**
  - Razmatraju se samo oni putevi na kojima modul nije beskonačno dugo u kritičnom odsječku. Naime, u modelu je (zbog nedeterminizma) moguće da pojedini modul ostane u svom kritičnom odsječku koliko god želi. To je trivijalan slučaj zbog kojeg može doći do kršenja svojstva životnosti.

# FSM za primjer MUTEX-a





# Jezik NuSMV korišten u stilu ograničenja (engl. *constraint*)

- U NuSMV-u moguće je osim stila dodjeljivanja (engl. *assignment*) koristiti i **stil ograničenja**:

*INIT* *<simp\_expr>* umjesto *init(atom) := <simp\_expr>*;  
*TRANS* *<next\_expr>* umjesto *next(atom) := <simp\_expr>*;  
*INVAR* *<simp\_expr>* umjesto *atom := <simp\_expr>*;

- Primjeri prevođenja stila dodjeljivanja u stil ograničenja:

*init(state) := {ready,busy};*    *INIT state in {ready,busy}*  
*next(state) := ready;*            *TRANS next(state) = ready*  
*out := b2 + 3\*b1;*                *INVAR out = b2 + 3\*b1*

- Ograničenjima zapravo navodimo logičke (*boolean*) tvrdnje koje trebaju vrijediti u modelu
- Svaki modul može imati nula, jedno ili više ograničenja, a moguće je i kombinirati dodjeljivanja i ograničenja

# Napomene oko korištenja stila ograničenja

- Rizik definiranja nekonzistentnih početnih ograničenja, npr.  
`INIT (p & !p) -- nepostojanje inicijalnog stanja - bilo koja specifikacija će biti zadovoljena`
- Rizik definiranja nekonzistentnih ograničenja na prijelaze
  - Relacija prijelaza Kripkeove strukture nije potpuna (iz nekog stanja nisu specificirani daljnji prijelazi), npr.  
`MODULE main`  
`VAR b : boolean;`  
`TRANS b = TRUE -> FALSE -- umjesto: ... -> next(b) = FALSE`
- Nedeterminizam je često skriven u ograničenjima, npr.  
`next(state) := case`  
`(state = ready & request = TRUE) : busy;`  
`TRUE : { ready, busy }; esac`  
`=>`  
`TRANS (state = ready & request = TRUE) -> next(state) = busy`

# Primjer: najveći zajednički djelitelj (1/3)

- Neka je zadan program u C-u:

```
void main() {  
    ...// inicijalizacija brojeva a i b, pretpostavka da su oba pozitivna  
    while (a!=b) {           // oznaka L1  
        if (a>b)              // oznaka L2  
            a=a-b;           // oznaka L3  
        else  
            b=b-a;           // oznaka L4  
    }  
    ...// NZD=a=b           // oznaka L5  
}
```

# Primjer: najveći zajednički djelitelj (2/3)

- Program preveden u NuSMV, u stil dodjeljivanja:

```
MODULE main()
VAR a: 0..100; b: 0..100;
pc: {L1,L2,L3,L4,L5};
ASSIGN
  init(pc) := L1;
  next(pc) := case
    pc=L1 & a!=b: L2;
    pc=L1 & a=b: L5;
    pc=L2 & a>b: L3;
    pc=L2 & a<=b: L4;
    pc=L3 | pc=L4: L1;
    pc=L5: L5;
  esac
...
next(a) := case
  pc=L3: a-b;
  TRUE: a;
esac
next(b) := case
  pc=L4: b-a;
  TRUE: b;
esac
...
```

# Primjer: najveći zajednički djelitelj (3/3)

- U NuSMV-u, stil ograničenja (primijetiti da više sličí originalnom programu)

```
MODULE main
```

```
VAR a : 0..100; b : 0..100; pc : {L1, L2, L3, L4, L5};
```

```
INIT pc = L1
```

```
TRANS
```

```
pc = L1-> ((a != b & next(pc) = L2) | (a = b & next(pc) = L5))  
          & next(a) = a & next(b) = b)
```

```
TRANS
```

```
pc = L2-> ((a > b & next(pc) = L3) | (a < b & next(pc) = L4))  
          & next(a) = a & next(b) = b)
```

```
TRANS
```

```
pc = L3 -> (next(pc) = L1 & next(a) = (a - b) & next(b) = b)
```

```
TRANS
```

```
pc = L4 -> (next(pc) = L1 & next(b) = (b - a) & next(a) = a)
```

```
TRANS
```

```
pc = L5 -> (next(pc) = L5 & next(a) = a & next(b) = b)
```

# Pokretanje sustava NuSMV

- NuSMV je moguće pokrenuti na dva načina: uobičajen (*default*) i interaktivan
- Uobičajen način:
  - Provjera svojstava svih CTL specifikacija zadanih u ulaznoj datoteci odjednom, nema mogućnost podešavanja
  - Sintaksa poziva: `> NuSMV <put_do_datoteke.smv>`
- Interaktivan način:
  - Omogućuje podešavanja parametara algoritma verifikacije, provjeru konačnog automata, simulaciju prolazaka kroz stanja procesa, provjeravanje jedne po jedne specifikacije...
  - Sintaksa poziva: `> NuSMV -int`
  - Učitavanje modela: `> read_model -i <put_do_datoteke.smv>`
- U okviru I. domaće zadaće studenti se upoznaju s oba načina rada, ali samo s jednostavnim funkcionalnostima interaktivnog načina

# I. zadatak

Za zadani model u NuSMV-u nacrtati Kripkeovu strukturu, te odrediti istinitost sljedećih CTL izraza:

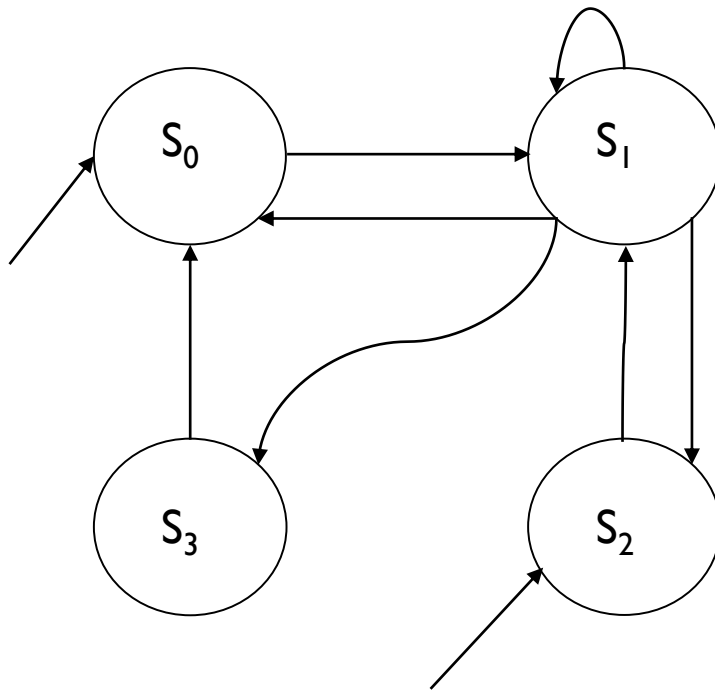
a)  $AG(AF(a \ \& \ b))$       b)  $EX(!a \ \& \ !b)$

```
MODULE mod
VAR a : boolean;
    b : boolean;
ASSIGN
    init(a) := FALSE;
    init(b) := FALSE;
    next(a) :=
        case
            (b = FALSE) : a;
            TRUE : !a;
        esac;
    next(b) :=
        case
            TRUE : !b;
        esac;
```

## 2. zadatak

Za zadanu Kripkeovu strukturu napisati kompletan NuSMV-kôd.

Pomoću ograničenja pravednosti osigurati da se sustav beskonačno često nađe u stanju  $S_2$



$S_0 = \{\text{req}=\text{FALSE}, \text{stat}=\text{FALSE}\}$

$S_1 = \{\text{req}=\text{FALSE}, \text{stat}=\text{TRUE}\}$

$S_2 = \{\text{req}=\text{TRUE}, \text{stat}=\text{FALSE}\}$

$S_3 = \{\text{req}=\text{TRUE}, \text{stat}=\text{TRUE}\}$



### 3. zadatak

Za zadani kôd modula u NuSMV-u potrebno je nacrtati odgovarajuću Kripkeovu strukturu i odrediti istinitost CTL specifikacija.

```
MODULE igrac_3
VAR
    ready : boolean;
    turn : {one, two, three};
ASSIGN
    init (ready) := TRUE;
    init (turn) := one;
    next (ready) :=
        case
            (!ready & (turn = one | turn = two)) : TRUE;
            (!ready & turn = three) : FALSE;
            TRUE : {FALSE,TRUE}; esac;
    next (turn) :=
        case
            (!ready & turn = one) : two;
            (!ready & turn = two) : {one,two,three};
            (ready & turn = one) : {one,two};
            TRUE: three; esac;
```

- A) CTLSPEC AF AG (turn = three)
- B) CTLSPEC EF AX ready

# Oko I. domaće zadaće

- **I. domaća zadaća uključuje**
  - Ispitivanje svojstava MUTEX-a na par različitih implementacija korištenjem specifikacija pisanih u CTL-u
  - Ispitivanje pravednosti istih implementacija
  - Provjera modela nekoliko različitih zadataka
  - Interaktivno pokretanje NuSMV-a i provjeru modela na nekoliko primjera
  - Nekoliko zadataka vezanih uz sustav NuSMV
- **Upute i rokovi za I. domaću zadaću dostupni su na web stranicama predmeta**